International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:5, 2007

A Real-Time Tracking System Developed for an Interactive Stage Performance

S. Hu, J. Mortensen and Bernard F. Buxton

***Abstract*—**A real-time tracking system was built to track performers on an interactive stage. Using an ordinary, up to date, desktop workstation, the performers' silhouette was segmented from the background and parameterized by calculating the normalized central image moments. In the stage system, the silhouette moments were then sent to a parallel workstation, which used them to generate corresponding 3D virtual geometry and projected the generated graphic back onto the stage.

***Keywords*—**image moment, interactive stage, real-time, silhouette.

I. Introduction

ARTISTS are interested in the integration the physical and virtual spaces in an interactive stage [1]. For example, in the *Spawn* project [2], a complex 3D geometry, which was composed of four circles stretching a spline-based membrane between them, deformed according to the performers' movements on stage. The 3D geometry was then projected in real-time onto a mobile screen back on the stage. The performers on stage and the visualization of the virtual geometry on the mobile screen embody the interaction between the physical and virtual spaces.

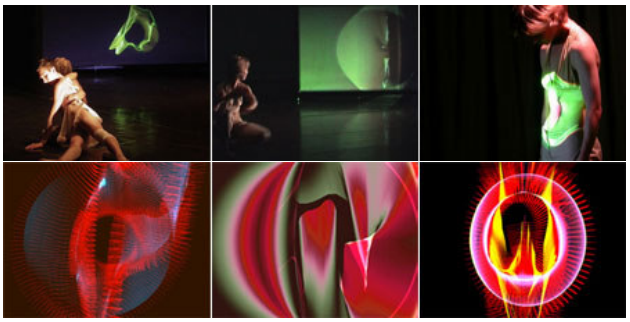


Fig 1 Pictures in the first row show the interactive stage and pictures in the second row show the deformable virtual geometry.

Manuscript received March 15, 2005

S. Hu. Author is with the Computer Science Department, University College London, Gower Street, London WC1E 6BT, UK (e-mail: s.hu@cs.ucl.ac.uk).

J. Mortensen. Author is with the Computer Science Department, University College London, Gower Street, London WC1E 6BT, UK (e-mail: j.mortensen@cs.ucl.ac.uk).

B. F. Buxton. Author is with the Computer Science Department, University College London, Gower Street, London WC1E 6BT, UK (e-mail: b.buxton@cs.ucl.ac.uk).

To enable the physical space and the virtual space merge in order to affect each other in such an interaction, in *Spawn*, we needed to coordinate visualization of the virtual geometry with the performers' movements. This required live tracking of the performers' movements, together with on-line generation of the corresponding 3D graphics, both running in real-time, continuously for up to 20-30 minutes.

In this paper we will introduce the tracking system that we built for the interactive stage in the *Spawn* project. The system captured the stage and the performers using an ordinary digital camcorder, which was placed besides the stage. The performers' silhouette (since the two dancers kept body contact during the whole performance, we regarded their connected silhouettes as a single one) was then segmented from the background. The silhouette's normalized central moments, which are invariant to image rotation and approximately invariant to changes in viewing distance, were then calculated to provide a low detail representation of the performers' body outline. In *Spawn*, the silhouette moments, formatted using the Extensible Markup Language (XML) [3], were then transmitted to a parallel workstation as input to generate the corresponding virtual geometry and finally projected back onto the screen on stage. The whole system, including the tracking system, has been implemented in real-time for real data on ordinary workstations (DELL dual P4 xeon 2.8ghz with 1GB RAM and a Geforce 6800) and shown to work for data obtained in the laboratory, an office environment, or a dance studio.

II. Silhouette Segmentation

In order to capture the performers' movements on stage, we need to segment the bodies' silhouette from the background. As a low computational cost algorithm is essential to the real-time system, we used a segmentation method based on background differencing [4] [5]

A. Learning the Background

During a training process when there were no performers on stage, the system learned what the background was on average and every background pixel's variation owing to illumination or other small changes. If there were N frames of background captured during the training process, then for each pixel (x, y) we calculated the sum of the pixel values, $SP(x, y)$, and their sum of squares, $SSP(x, y)$, from which the mean pixel value,

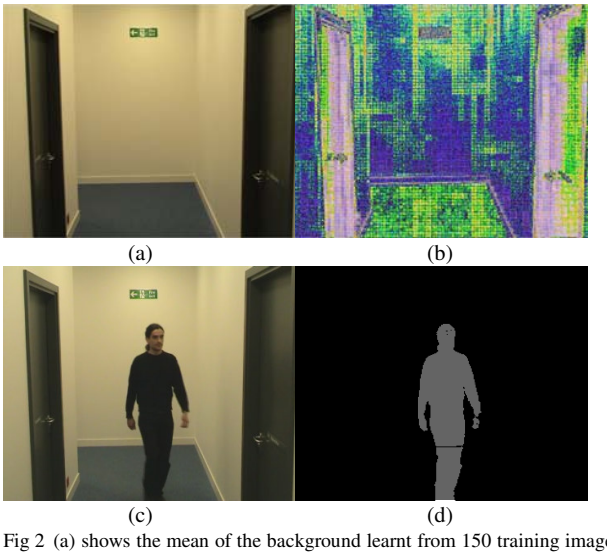
$M(x, y) = SP(x, y) / N$, and the standard deviation $\sigma(x, y)$ were calculated.

B. Extract the Foreground

We assume that pixels of the background may change their intensities independently, according to a normal distribution, for example owing to noise or other random effects that we do not wish to model in detail. For every pixel in a grey scale image, we could then use the standard deviation $\sigma(x, y)$ of that pixel obtained from the learning phase to judge whether the pixel belongs to the foreground or not. Thus if

$$abs\big(M(x, y) - I(x, y)\big) \geq c\sigma(x, y), \quad (1)$$

where $I(x, y)$ is the current intensity of pixel (x, y) and c is a constant value that would typically be set approximately to 3, we can say that the pixel located at (x, y) belongs to a foreground object. For colour images, as captured in the interactive stage application, equation (1) was applied to each colour channel (red, green, blue), according to the means and standard deviations as calculated for each channel, as in [4]. In our applications, since we assumed an independent distribution on each pixel, it was not necessary to use a mixture model. However, in order to reduce the number of false positive pixels, we adopted the conservative strategy that only pixels fulfilling (1) on *every* colour channel were regarded as belonging to the performers' silhouette. This approach has been tested on data obtained in the laboratory and an office environment as shown in Fig 2.



(a) (b)

(c) (d)

Fig 2 (a) shows the mean of the background learnt from 150 training images. (b) shows the standard deviation of the background pixels. (c) shows a person walking into the scene, and (d) shows the segmented silhouette of the person obtained by using the method described in section II with c set to a high value of 5, in this case to avoid false positive pixels on the door frames.

C. Shadow Elimination

For the method discussed in the previous section, we often mislabel pixels in the shadow of a performer as belonging to the foreground. Because the shadows are normally connected to the performers' bodies and may cover an area larger than the object itself, it is important to eliminate such shadows, yet difficult to do so by conventional methods such as morphological filtering or by selecting the largest area or longest contour. To tackle this problem, we introduced a method that is based on chromaticity information as follows in a manner similar to, but simpler than that of Bowden [6].

Chromaticity is a representation of colour that is independent of intensity. If we assume the shadows are caused by part of the background receiving less light when the object moved in the scene, then we may assume that each background pixel has similar chromaticity properties before and after being covered by a shadow. For pixels covered by the object, however, their chromaticity properties will change significantly if the object's colour is different from that of the part of the background that is covered.

For the pixel (x, y) , chromaticity can be obtained by colour normalization, with for example, for the red channel:

$$C(r; x, y) = \frac{I(r; x, y)}{I(r; x, y) + I(g; x, y) + I(b; x, y)}, \quad (2)$$

with similar equations for the green and blue channels, where $C(x, y) = \big(C(r; x, y), C(g; x, y), C(b; x, y)\big)$ stands for the chromaticity of pixel (x, y) and $I(r; x, y)$, $I(g; x, y)$, and $I(b; x, y)$ are the intensities of the red, green and blue colour channels respectively. Then, for pixel (x, y) , the difference $\Delta C(x, y)$ between its chromaticity $C'(x, y)$ when given a new image frame and that $\overline{C}(x, y)$ learnt during the background training process can be measured by the Mahalanobis distance $d(x, y)$ defined formally as follows:

$$d(x, y) = \sqrt{\Delta C(x, y)^T S^{-1}(x, y) \Delta C(x, y)}, \quad (3)$$

where $S(x, y)$ is the chromaticity covariance matrix of pixel (x, y) , also obtained during the background training process.

Then, if for a new frame, for each pixel, the distance $d(x, y)$ is bigger than a constant (again in the range 3-5), the pixel will be regarded as belonging to the foreground, otherwise it will be regarded as belonging to the background .

Since, in the RGB space, $C(x, y)$ lies on the 2D chromaticity plane, the 3×3 matrix $S(x, y)$ will be singular, so the $S^{-1}(x, y)$ in equation (3) must be treated as a pseudoinverse. This may conveniently be computed from the principal components of the covariance matrix $S(x, y)$ and the distance $d(x, y)$ calculated in practice as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^{2} \frac{\Delta C(x, y)^T \, p(i; x, y) \, p(i; x, y)^T \, \Delta C(x, y)}{\lambda(i; x, y)}}, \quad (4)$$

where $\lambda(i; x, y)$ is the i^{th} biggest eigenvalue of $S(x, y)$ and $p(i; x, y)$ is the corresponding eigenvector.

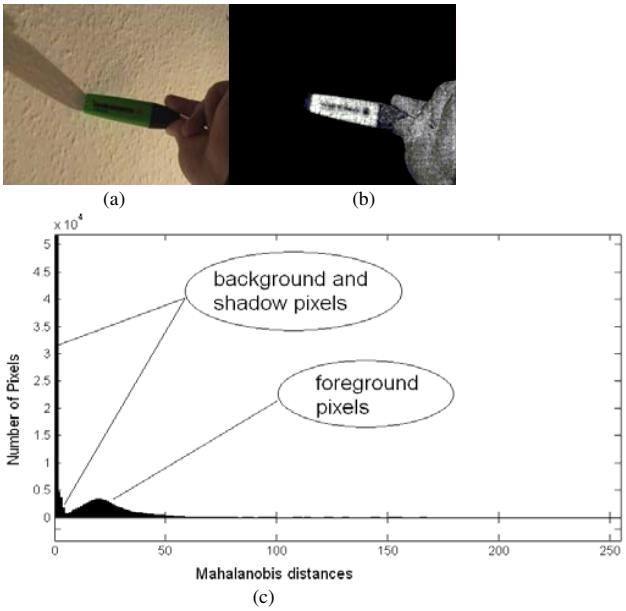


Fig 3 100 images of the wall (light yellow) were used to learn the background. (a) shows objects with different colours (the hand and the marker pen) moved into the scene. (b) shows the Mahalanobis distance for each pixel calculated using equation (4) with all the distances multiplied by 5 to make the image visible. It can be seen that the distances in the shadow region are smaller than those in the area covered by the objects. (c) shows the histogram of Mahalanobis distances.

Fig 3 above shows the Mahalanobis distances calculated for an example image. It can be seen from the histogram that most of the pixels that are not covered by the objects are at zero or small Mahalanobis distances from the background, whilst pixels covered by the objects are further from the background at larger distances. More results are shown in the Implementation section.

D. Computing the Silhouette Moments

After the performers' silhouette were obtained, we calculated the silhouette's normalized central moments, which may be defined over the R. By definition, the zero order normalised moment is one and the first order central moments vanish.

Calculating the moments over the R would require a significant amount of computation proportional to the number of pixels within R. However, according to Green's theorem [7], instead of computing moments over the region R, we can compute the moments along the contour of R and hence involve fewer pixels and save computational effort.

III. Implementation

The tracking system consisted of one workstation running under the Windows XP operating system and one ordinary digital camcorder, which was mounted on a tripod about 1.2 meters high. The camcorder and the workstation were connected with an IEEE 1394/FireWire cable. The workstation of the tracking system was then connected, via an Ethernet link, to a parallel similar workstation which was used to generate the virtual graphics.

A. System architecture

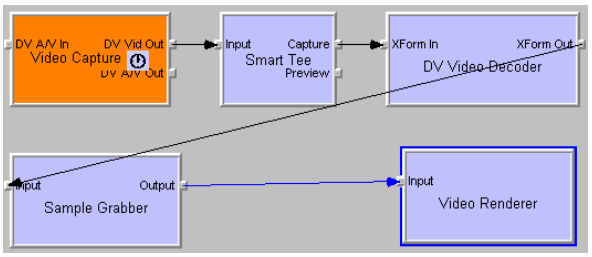


Fig 4 Architecture of the tracking system

The tracking system's software program, which controlled the camcorder and performed silhouette moment calculation, was developed under C++ and compiled using Microsoft Visual Studio .NET 2003.

To make the capture easy and efficient under a Windows XP environment, we used Microsoft DirectShow, which provides a set of low-level application programming interfaces (APIs) under Microsoft DirectX, with which to build the system as illustrated schematically in Fig 4.

Five different filters from DirectShow were used in the tracking pipeline. The first, at the top left in Fig 4, was a Windows Driver Model (WDM) video capture filter. This filter was the interface between the capture device (a digital camcorder) and the program. It took the input signal from the camcorder at 25 frames per second, split the signal into an audio stream and a video stream and, in the system developed for Spawn, passed the video stream on to the next filter. In our system, the audio was not used.

Without any additional data copying, the second filter (Smart Tee) split the video stream into a preview stream and a capture stream. However, to avoid imposing an unnecessary burden on the system, we didn't record or display the preview stream except for development purposes and, in a performance, only the capture stream was passed down the pipeline.

The capture stream from the Smart Tee filter was still in digital video (DV) format, which is not convenient for processing. A DV Video Decoder was thus inserted to convert the capture stream in to an uncompressed video format so that each frame could be processed as ordinary RGB color imagery.

After inserting a Sample Grabber filter as show in the Fig 4, we had two options for retrieval of captured frames for processing. One was to buffer each frame the Sample Grabber filter received; the other was to use a callback function. Invoking a callback function may cause deadlocks, which are unacceptable during a live performance. Since this may happen whenever the computation cost rises significantly, for example, on frames when the performers move too close to the camera and covers most of the camera's view, we chose to use

the buffering approach. Each frame we buffered was processed as discussed in section II and the silhouette moments were sent to the parallel workstation before the next buffer was retrieved. We used a Video Render filter to end the processing pipe and to monitor the captured frames.

B. Silhouette Segmentation, Shadow Elimination and Moment Calculation

For the frame buffers obtained from the Sample Grabber filter, we employed an open source library called OpenCV [8] to perform the silhouette segmentation and the moment calculations. The OpenCV library provided efficient and quick access to each pixel in the frame, which helped us identify the pixels that exceed the allowed variance $c\sigma(x, y)$ as in (1). We labelled these pixels as possible foreground pixels. Binary images were created by setting the foreground pixels to 1 and others pixels to 0. A median filter with a 5x5 pixel mask was applied to remove salt and pepper noise by using the OpenCV function "cvSmooth".

The shadow elimination method defined by equation (4) was then applied to every pixel (x, y) which was still labeled after the smoothing as 1 in the binary images. This enabled us to remove most shadow pixels that were mislabelled as foreground in the binary image.

Then, by using functions like "CvContourScanner" of OpenCV, we obtained the contours of the foreground pixel regions. Usually, we obtained more than one contour at this stage because some noise may pass the median filter and the shadow elimination processes. We solved this problem by checking each contour's length and keeping only the longest contour, which it is reasonable to assume defines the outline of the performers' bodies. Finally, the silhouette moments were calculated by using the function "cvMoments", which takes the contour as input. Note that although we sent silhouette moments to the parallel workstation, there is no reason why, in principle, we can not send the contour together with the moments as input to generate the virtual geometry.

Some experiments on this tracking system were carried out in the laboratory as shown in Fig 5. It can be seen that the system gave good results except when we used the yellow marker pen, which has a similar colour to the yellow background and, as expected, made the shadow elimination process fail by mislabelling object pixels as background.

IV. Conclusion

In this paper we have introduced a real-time tracking system, which was built on an ordinary workstation running under the Windows XP operating system. The tracking system was used to track a moving object before a steady background, to extract its silhouette, and to send normalised central moments computed from the silhouette to another workstation as input for generating the virtual geometry. The whole system has been tested in a laboratory and in an office environment and used during online during a live, stage performance. Satisfactory results were obtained throughout, and the system ran continuously in real-time without trouble throughout several 20-30 minute dance routines.

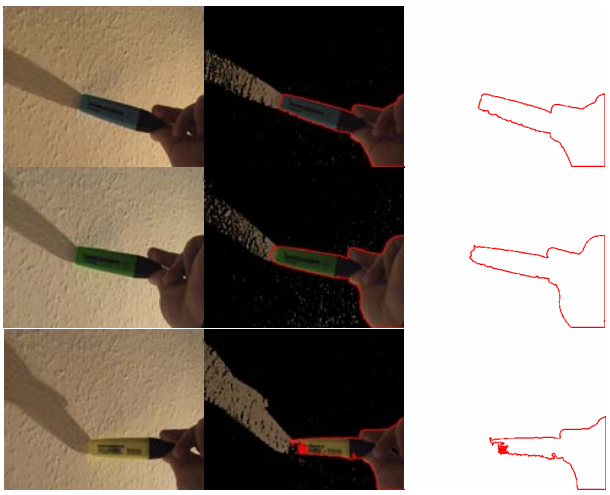


Fig 5 The first column of pictures on the left shows the test frame grabbed from the Sample Grabber filter, the second column in the middle shows the segmentation results. All pixels that failed to fulfill equation (1) were set as 0 (black in image). It can be seen in the second column that many shadow pixels passed equation (1). After shadow elimination, the final contours are shown in the third column on the right. It can be seen that the tracking system gives good result on the blue maker pen (top row), green marker pen (middle row) and the hand while mislabelled some pixels of the yellow marker pen (bottom row).

Acknowledgment

The authors would like to thank Mette Thomsen and Carol Brown for suggesting the project to them, Chris Parker asnd Mette Thomsen for their help and support, and the dancers of Carol Brown Dancers for their performances.

References

[1] R. Bowden, P. KaewTraKulPong and M. Lewin, *Jeremiah: The Face of Computer Vision*, Smart Graphics'02. Conf. Proc. Series, page 124-128, June 2002, Hawthorn NY USA,

[2] Mette Thomsen, *Spawn project*, [Online] Available at http://www.cs.ucl.ac.uk/research/vr/Projects/VLF/Media/escape/spawn.html, (accessed 03 March, 2005).

[3] *Extensible Markup Language* (XML), [Online] Available at http://www.w3.org/XML/, (accessed 10 March, 2005)

[4] A. Elgammal, D Harwood and L. Davis, *Non-Parametric Model for Background Subtraction,* ECCV2000, volume 2, page 751–767, 2000.

[5] C.R.Wern, A. Azarbayejani, T. Darrell and A. P. Pentland, *Pfinder: Real-time tracking of human body,* IEEE Trans on Pattern Analysis and Machine Intelligence, 1997.

[6] P. AaewTraKulPong and R. Bowden, *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection,* In: Proc. AVBS'01, 2001.

[7] C. H. Edwards, Jr., and D. E. Penny, Calculus and analytic geometry, 3rd Edition, page 859-866, 1982, Prentic Hall.

[8] *The Open Computer Vision Library*, [Online] Available at http://sourceforge.net/projects/opencvlibrary/, (accessed 10 March, 2005)